Integrating GEANT4 with Athena, the ATLAS software framework

GEANT4 User's Workshop

Feb 21 2002

User's Session

Charles Leggett

Lawrence Berkeley National Lab

Athena

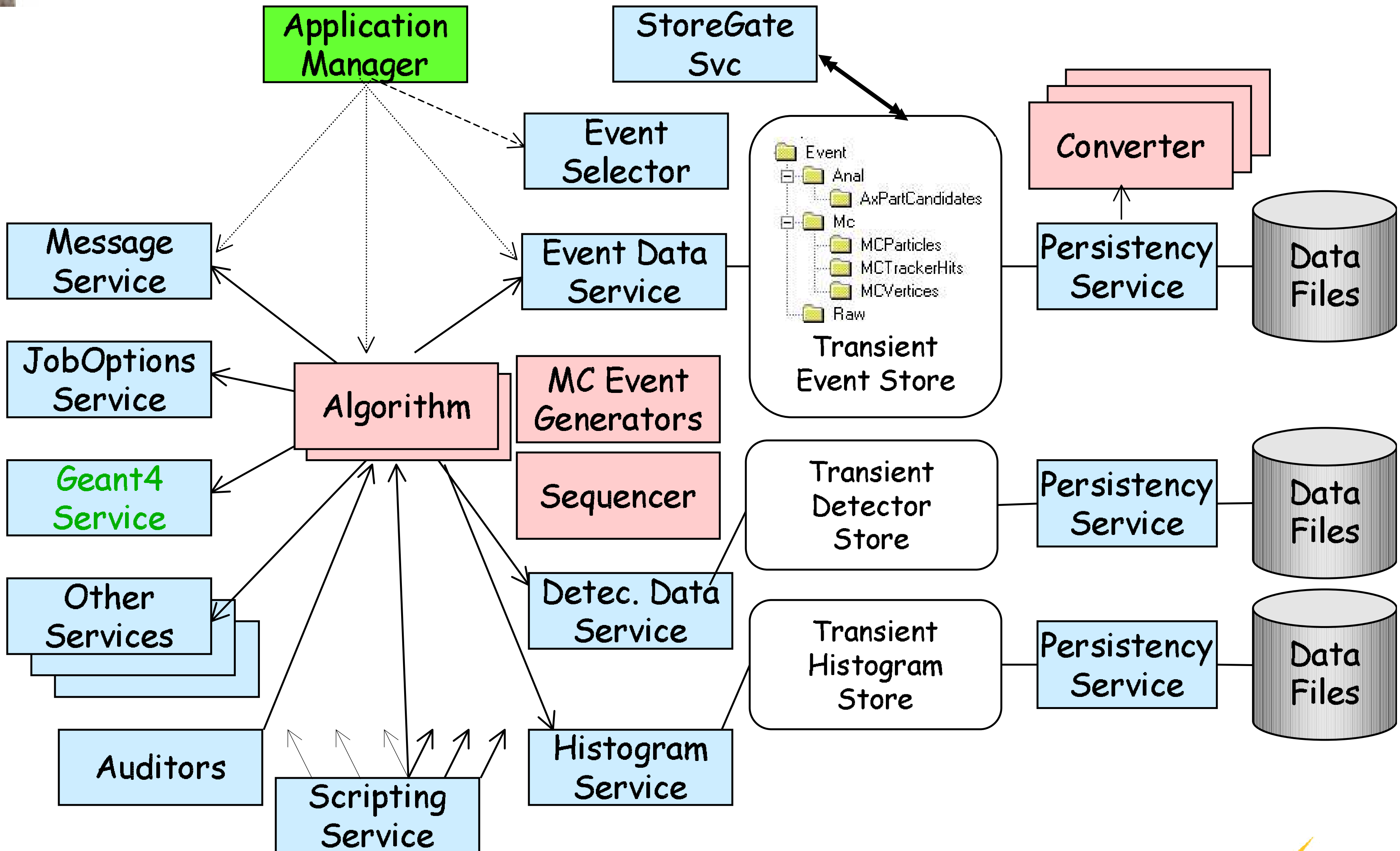
- It's a framework:
  - Represents a collection of classes that provide a set of services for a particular domain.
  - A skeleton of an application into which developers plug in their code and provides most of the common functionality.
- Utilizes components:
  - A physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces.
- Framework Design Goals:
  - Object oriented paradigm
    - C++ implementation language
    - Java foreseen
  - Separation of Data and Algorithms
  - Separation of Transient and Persistent Data
    - Independence from persistent implementation

Athena

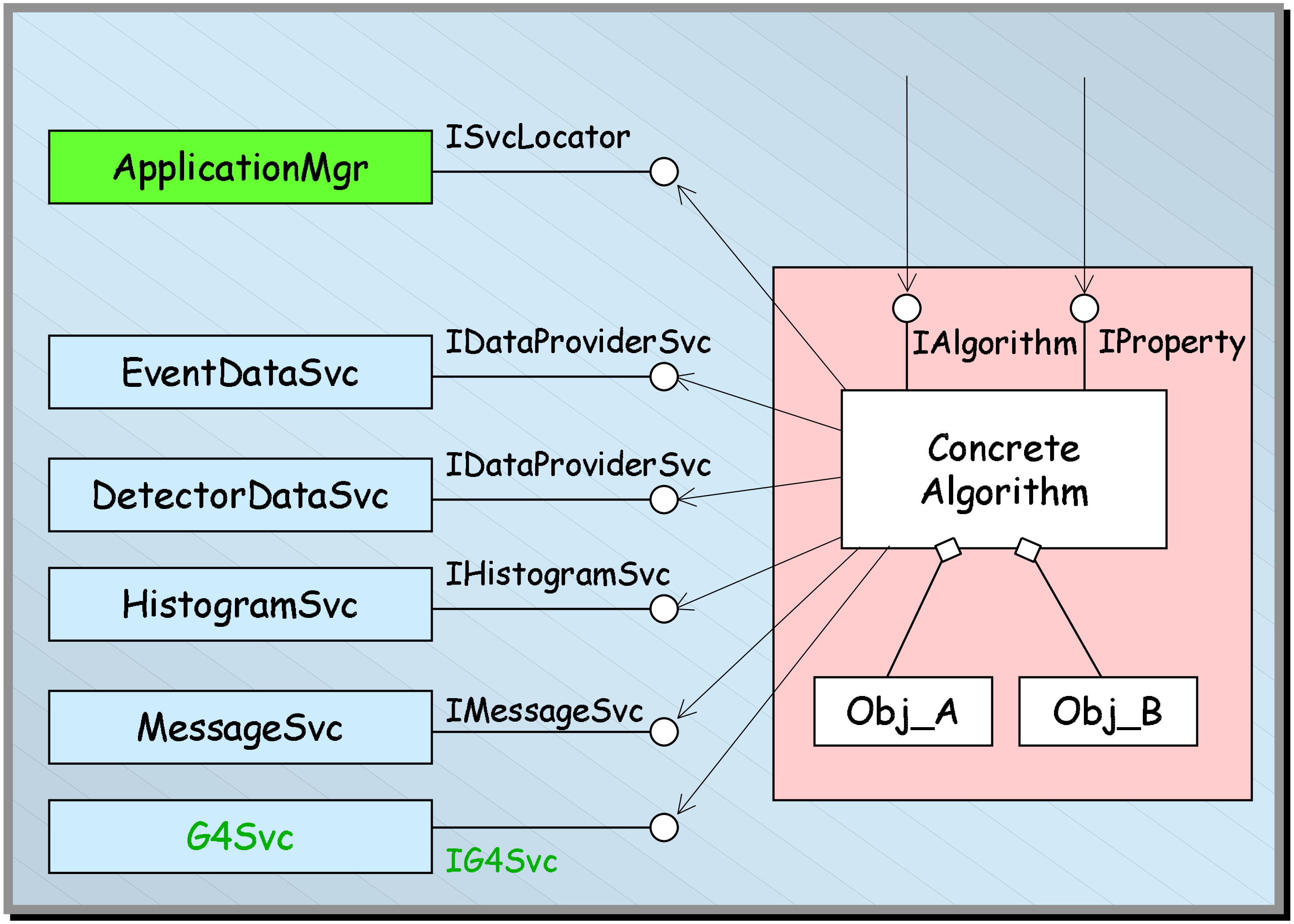
- Based on the LHCB framework: **Gaudi**
- Slightly customized: extends or replaces several core services

- Core abstractions:
  - Algorithms: computational code
  - Data Objects: transient objects capable of being converted
  - Converters: convert data from one representation to another
    - transient $\leftrightarrow$ persistent
    - transient $\rightarrow$ graphical
  - Services: components that provide a support service
    - histogram service
    - montecarlo generators
  - Data Stores: several, both transient and persistent

Athena-Gaudi Object Diagram



Interfaces



ApplicationMgr
ISvcLocator
EventDataSvc
IDataProviderSvc
DetectorDataSvc
IDataProviderSvc
HistogramSvc
IHistogramSvc
MessageSvc
IMessageSvc
G4Svc
IG4Svc
IAlgorithm
IProperty
Concrete Algorithm
Obj_A
Obj_B

Algorithm

- Users write concrete Algorithms derived from base class Algorithm
  - called once per physics event
  - Implement- at least - three methods in addition to the constructor and destructor
    - initialize() - called once at beginning of job
    - execute() - called for every event
    - finalize() - called once at end of job
- Algorithm provides hooks to common services
  - Other registered services are accessible once their header file is included in the Algorithm
  - Services are accessible via their abstract interfaces

Integration Objectives

- Access GEANT4 services from Athena framework
  - simple access for standard features
  - ability to use low level G4 functionality
- Use HepMC event as produced by Generators
- Understand various ATLAS geometry formats:
  - basic GEANT4 C++ geometry classes
  - multiple flavours of XML
- Use GEANT4 Physics Lists
- Preserve GEANT4 Hits and Tracks/Trajectories for further processing
- Visualization
- Use Athena framework facilities for persistification
- Use Athena framework facilities for histograms
- Use any other Athena framework component…

Components

- G4Svc: Athena/GAUDI service
  - G4Svc
    - inherits from IService, IG4Svc
    - usually, only thing the user needs to talk to
    - triggers G4SvcRunManager initialization, event processing loop and termination
  - G4SvcRunManager
    - inherits from G4RunManager
    - extends functionality
    - splits up event loop so G4 event/hit store doesn't get cleared until end of Athena event
    - no BeamOn method
  - AthenaHepMCtoG4EventAction
    - converts HepMC::GenEvent MonteCarlo event in Transient Event Store into a G4Event
- XML geometry builders
  - reads XML files and builds detector geometry and materials specifications

Access to GEANT4 Services

- Significant code already exists in ATLAS that uses G4 in a standalone capacity. Need to simplify transition to Athena for these users.

- Pure GEANT4 facilities can be accessed in several different ways:
  - using pre-defined G4Svc access methods, as defined in the Abstract Interface (IG4Svc), such as:
    - `SetUserAction( G4VUserEventAction );`
    - `SetUserInitialization( G4VUserDetectorConstruction* );`
  - direct access to G4RunManager
  - via command line userInterface session
  - sending text commands to the UImanager
  - get hold of various other G4 objects, such as G4Event* and talk directly to them

MC Event Conversion

- Convert HepMC::GenEvent as produced by Generators into a G4Event, using a class that inherits from G4VUserPrimaryGeneratorAction
- Can do multiple events per event (pileup)

- Trivial to do with SingleParticleGun

- For Pythia events, don't need all the particles, can purge unstable ones
- Problem: particle tree becomes disconnected.
  - put all particles into one primary vertex.
  - create many primary vertices, one for each HepMC::GenVertex
  - relink tree properly

Physics Lists

- Very simple: copy directly from GEANT4 examples.

- Select via jobOption
  - G4Svc.PhysicsList = “string”

- Available lists:
  - Geantino (ExN01, ExN02)
  - ElectroMagnetic (ExN03)
  - Full (ExN04)
  - “none” → user must supply their own

- Users can also create their own lists, and load them via
  - G4Svc->SetUserInitialization(PhysicsList*)

Process Loop: initialize()

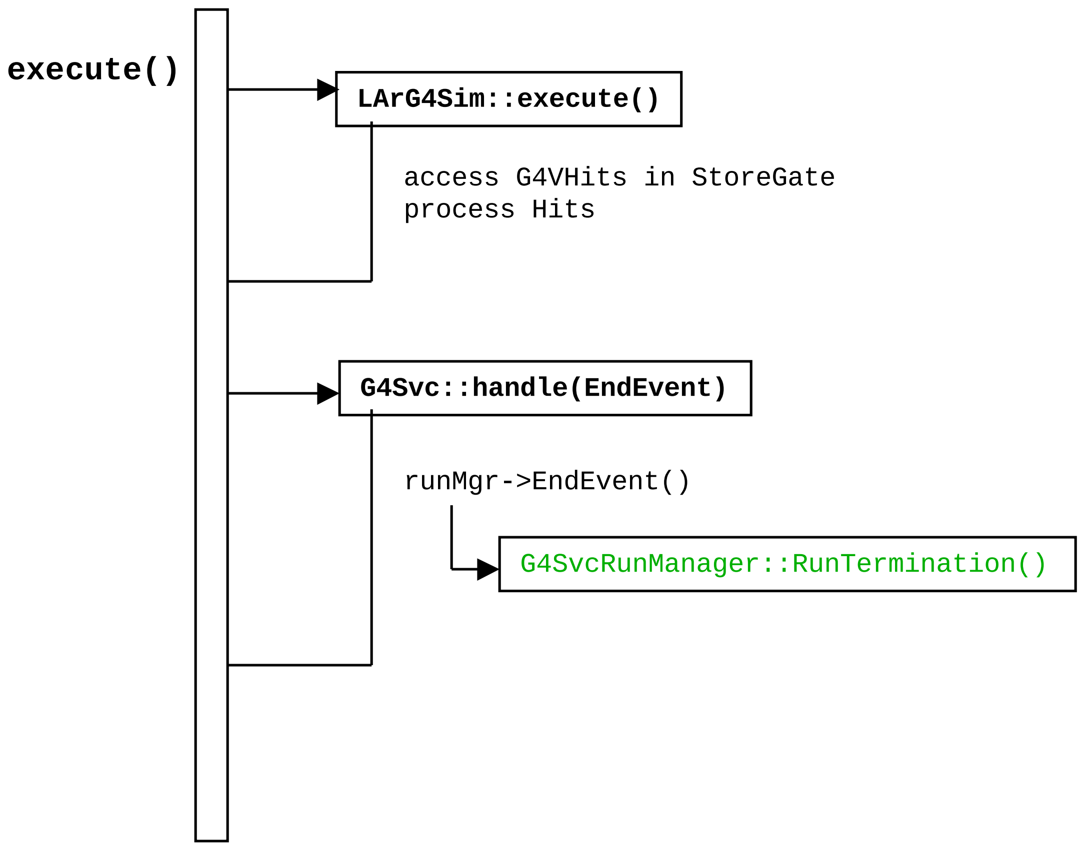
```
initialize()
    G4Svc::initialize()
        AddListner(EndEvent, McEventGenerated)
        new G4SvcRunManager

    LArG4Sim::initialize()
        G4Svc->SetUserInitialization(new LArDetectorConstruction)
        G4Svc->SetUserAction(new LArRunAction)
        G4Svc->SetUserAction(new LArEventAction)
        G4Svc->SetUserAction(new LArSteppingAction)
```

Process Loop: execute()

```
execute()
  GenModule::execute()
    create HepMCEventCollection, put in StoreGate
    fire McEventGenerated incident
      G4Svc::handle(McEventGenerated)
        if (FirstEvent) {
          runMgr->SetUserInitialization(PhysicsList)
          runMgr->SetUserAction(AthenaHepMCtoG4EventAction)
          setup visualization
          runMgr->Initialize()
        }
        G4SvcRunManager::StartEvent(n)
          SetNumberOfEventsToBeStored(n)
          RunInitialization()
          DoEventLoop(n)
            AthenaHepMCtoG4EventAction::GeneratePrimaries()
          for (1,n) {
            SaveHits()
            SaveTracks()
          }
```

Process Loop: execute()

cont.



Saving G4VHits

```
G4SvcRunManager::SaveHits(G4Event* event) {
  event->GetHCofThisEvent()
  new vector<string> hit_keys
  for (HitCollections in event) {
    new vector<G4VHit*> v_HC
    hit_keys->push_back(HitCollection->GetName())
    for (G4VHits in HitCollection) {
      v_HC->push_back(G4VHit)
    }
    StoreGate->record(v_HC,HitCollectionName)
  }
  StoreGate->record(hit_keys,"HitKeys")
}
```

Saving G4VTrajectories

- Very similar to G4VHits, except all stored in one container

```
G4SvcRunManager::SaveTracks(G4Event* event) {

  vector<G4VTrajectory*> *v_TC = new vector<G4VTrajectory*>;
  G4TrajectoryContainer* TC = event->GetTrajectoryContainer();

  for (G4VTrajectory* in TrajectoryContainer) {
      v_TC->push_back(G4VTrajectory*)
  }

  storeGate->record(v_TC,"G4VTrajectory")
}
```

- debug mode prints out all G4TrajectoryPoints in each G4VTrajectory. Turn off when using Pythia!

G4Svc Job Options

- G4Svc.PhysicsList:
  - "ExN01", "ExN02", "ExN03", "ExN04 "
  - "Geantino", "EM", "Full", "none"
- G4Svc.DefaultPhysicsCut
  - for a PhysicsList
- G4Svc.Visualize
  - turn on visualization
- G4Svc.VisType
  - default VRML
- G4Svc.SaveTracks, SaveHits
  - save G4Hits and G4Trajectories
- G4Svc.RunVerbosity, EventVerbosity, TrackingVerbosity
  - how much G4 output to print out

Accessing the G4Svc

- Tell the jobOptions about it:

```
ApplicationMgr.DLLs   += { "G4Svc" };
ApplicationMgr.ExtSvc += { "G4Svc" };
```

- Get hold of the service inside an Algorithm:

```
#include "G4Svc/IG4Svc.h"

IG4Svc* p_G4Svc;
StatusCode status = service("G4Svc",p_G4Svc);
```

Building the Geometry

- Users can create standard G4 C++ classes, and register them directly with the G4Svc

```
G4Svc->SetUserInitialization( new MyDetectorConstruction );
```

- ATLAS also uses XML to describe the detector geometry and materials. Several different competing models exist - a frustrating lack of standards. Generic geometry builders have been designed that will assemble a G4 geometry by parsing the XML.
  - packaged as separate Algorithms that build the geometry in their intialize() method
  - selected at run time by providing the appropriate jobOptions

Building Geometry from XML

- Several XML geom/material builders have been implemented:
  - Stan's G4Builder: G4Builder

```
ApplicationMgr.DLLs += { "G4Builder" };
ApplicationMgr.TopAlg = { ... ,  "G4BuilderAlg",  ... };

G4BuilderAlg.MaterialXML = "Material_AGDD.xml";
G4BuilderAlg.DetectorXML = "Atlas_AGDD.xml";
```

  - Jean-Francois's AGDDBuilder: G4AGDDBuilder

```
ApplicationMgr.DLLs += { "G4AGDDBuilder" };
ApplicationMgr.TopAlg = { ... ,  "G4AGDDBuilder",  ... };

G4AGDDBuilder.MaterialXML = "Material_AGDD.xml";
G4AGDDBuilder.DetectorXML = "Atlas_AGDD.xml";
```

  - Andrea's DOM model: G4DOMBuilder

```
ApplicationMgr.DLLs += { “G4DOMBuilder” };
ApplicationMgr.TopAlg = { ... ,  "G4DOMBuilder",  ... };

G4DOMBuilder.XML = { ”SCTDesc.xml”, “color.xml” };
```

Building AGDD Geometry

- Use AGDD to read in (uncompacted) XML

```
G4VPhysicalVolume* DetectorConstruction::Construct() {

  AGDD_Factory &f = AGDD_Factory::Expat_instance();
  f.build_detector_description (m_materialFile);

  BuildMaterial build_material;
  build_material.parseAGDD (f.get_detector_description() );

  f.build_detector_description (m_detectorFile);

  BuildGeometry build_geometry;
  build_geometry.parseAGDD (f.get_detector_description() );
}
```

G4Svc UserInterface

- Can access the G4UI text user interface at any time:
  - `p_G4Svc->StartUISession();`

- Can also pass commands directly to the UImanager:
  - `p_G4Svc->uiMgr()->ApplyCommand( "G4 command" );`

Accessing Hits and Tracks

- G4VHits and G4VTrajectories are stored in StoreGate in vectors, keyed by name
- In an Algorithm's execute method, to retrieve hits:

```
const DataHandle< vector<string> > hit_keys;
vector<string>::const_iterator kitr;

StatusCode status = m_sgSvc->retrieve( hit_keys, "HitKeys" );

if (status.isSuccess()) {

   for (kitr=hit_keys->begin(); kitr!=hit_keys->end(); ++kitr) {
      const DataHandle< vector<G4VHit*> > hits;
      status = m_sgSvc->retrieve(hits, (*kitr));

      if (status.isSuccess()) {
         vector<G4VHit*>::const_iterator itr;
         for (itr=hits->begin(); itr!=hits->end(); ++itr) {
            (*itr) -> Print();
         }
      }
   }
}
```

Visualization

- Several visualization models are currently supported:
  - VRML/vrweb
  - DAWN
  - will add the other G4 standards when I get the chance: had trouble compiling them in under Linux when I first started

- By turning visualization on via the jobOptions, the G4Svc will write out the appropriate visualization file at the end of every event.

Implementations

- GEANT4 novice examples 1 through 4
  - package G4Sim/G4Ex.
  - use Algorithm "G4ExN01", "G4ExN02", "G4ExN03", "G4ExN04"
- Liquid Argon Calorimeter,
  - package G4Sim/LArG4Sim
  - uses C++ classes
- SCT with G4Builder
  - package G4Sim/G4Builder
- SCT,Muon,HEC with G4AGDDBuilder
  - package G4Sim/G4AGDDBuilder
- SCT with G4DOM
  - package G4Sim/G4DOMBuilder

Status

- Tested with G4.3.2, G4.4.0
- Survived tens of thousands of single particle events, and a thousand+ of Pythia events

- Still in prototype phase
- Waiting to hear back from users as to desired features, and undesired features (bug reports)

- We hope to use it in the ATLAS Data Challenge, part 1, phase B

General Comments

- Not entirely trivial to use Geant4 in "non-standard" ways.
- In many cases it was difficult to get to the guts of the various managers. Instead of providing accessors to functionality, had to send text commands with the ApplyCommand() method of the G4UImanager.
- Had to hack the RunManager and split it into two parts to prevent G4 from deleting the Hits/Tracks before I was ready to use them.

- Compiling/linking/running tricky since didn't know what libraries to include. Need a "geant4-config" à la "root-config" or "cernlib"

Documentation

- online:
  - http://annwm.lbl.gov/G4
- CVS:
  - ATLAS CVS repository
  - $CVSROOT = :kserver:atlas-sw.cern.ch:/atlascvs
  - offline/Simulation/G4Sim/G4Svc